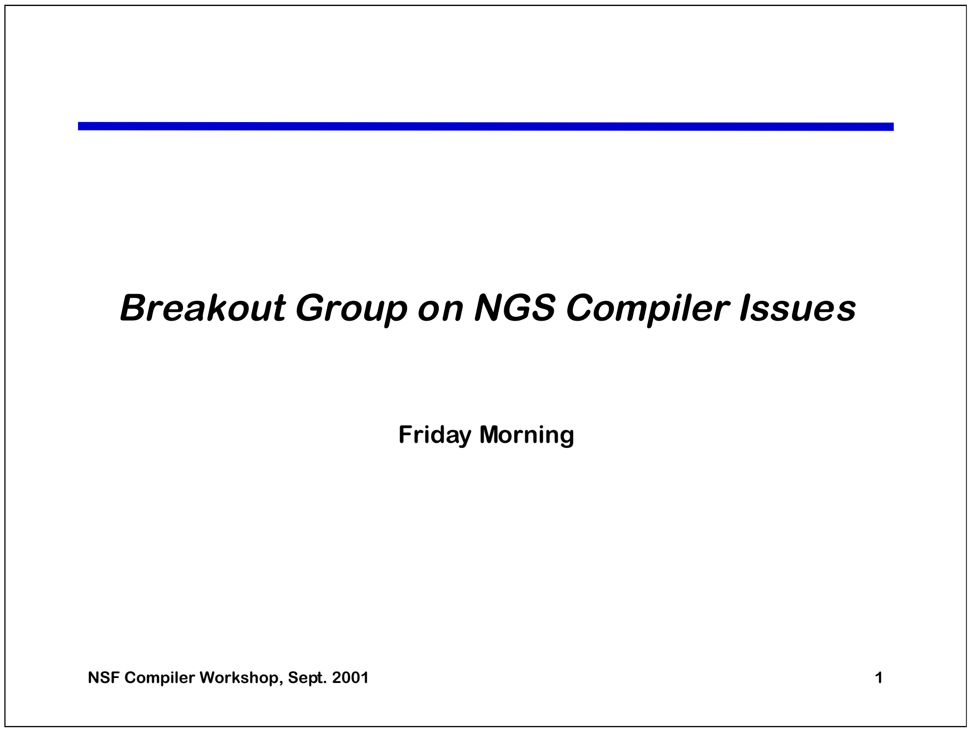


Breakout Group on NGS Compiler Issues

Friday Morning

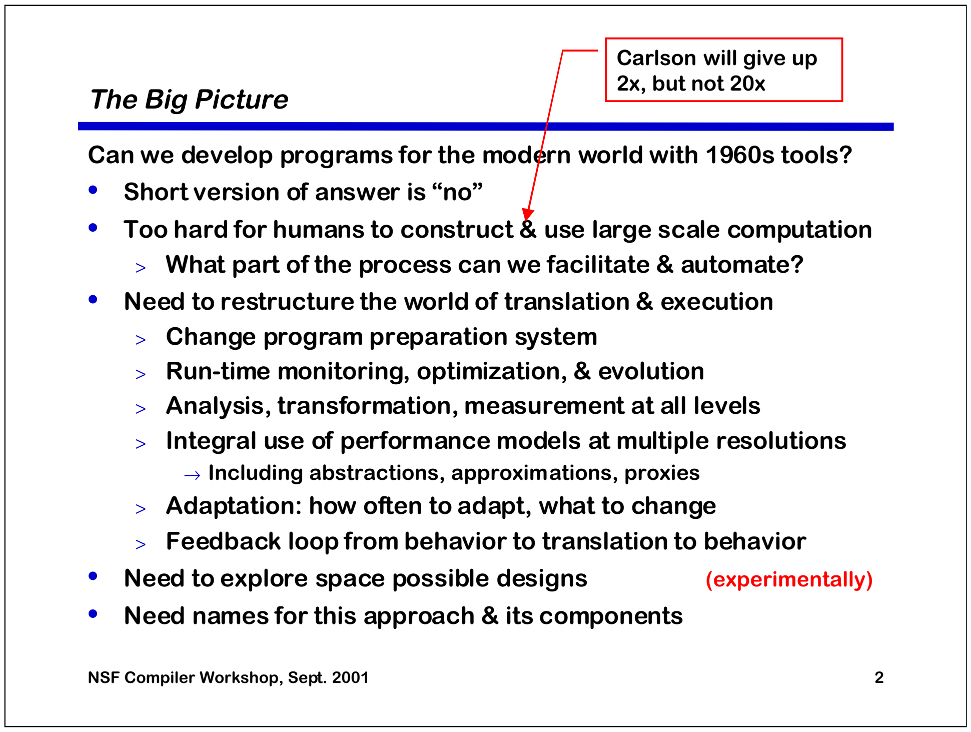
I wasn't the scribe, so I didn't get a full participants list.

From memory, the group included

Bill Carlson, Jacl Davidson, Mary Hall, Elliott Moss, Barbara Ryder Mike Smith, Mary Lou Soffa, Chip Weens and Keith Cooper

I feel certain that I have left out one or more people

This file contains two sets of slides. The first set is a slightly updated version of the summary talk that I gave Friday afternoon. The second set records the contents of the Flip Charts that Chip Weems made during our deliberations.



Carlson will give up 2x, but not 20x

The Big Picture

Can we develop programs for the modern world with 1960s tools?

- Short version of answer is “no”
- Too hard for humans to construct & use large scale computation
  - > What part of the process can we facilitate & automate?
- Need to restructure the world of translation & execution
  - > Change program preparation system
  - > Run-time monitoring, optimization, & evolution
  - > Analysis, transformation, measurement at all levels
  - > Integral use of performance models at multiple resolutions
    - → Including abstractions, approximations, proxies
  - > Adaptation: how often to adapt, what to change
  - > Feedback loop from behavior to translation to behavior
- Need to explore space possible designs (experimentally)
- Need names for this approach & its components

NSF Compiler Workshop, Sept. 2001 2

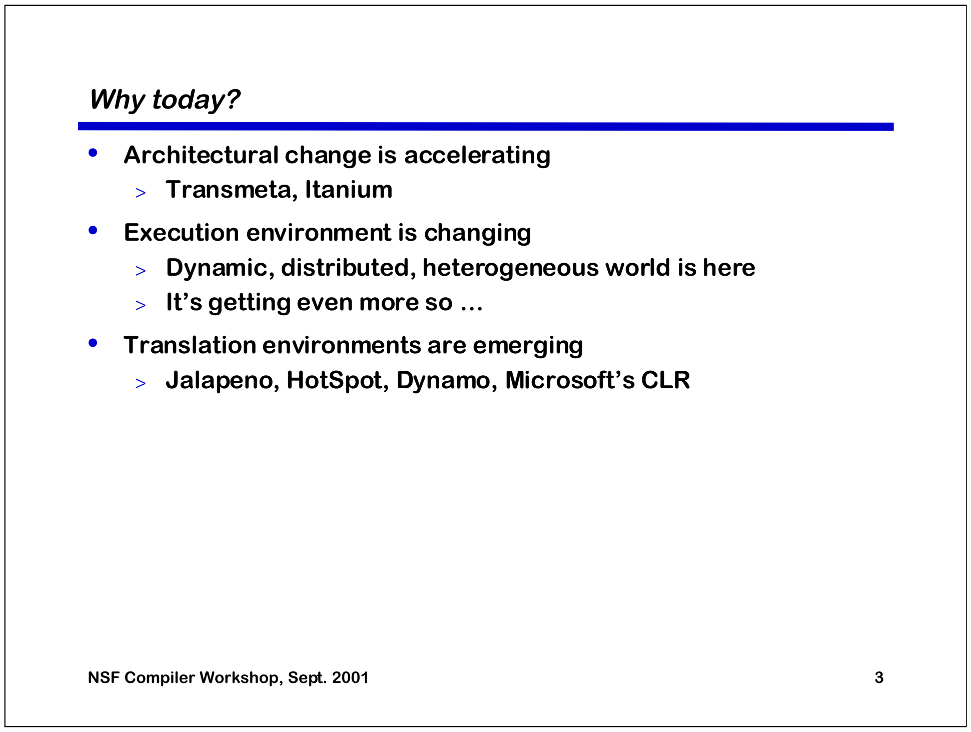
Carlson noted that a major impediment to large scale computation is the difficulty of building, running, and understanding these codes.

To make such activity easier, we need to rethink the entire process.

<editorial remark> This undoubtedly requires that we take techniques developed for use in compilers (at all levels) and apply them elsewhere in the program development and execution cycle. For example, dynamic compilation and optimization moves techniques from the static compiler into the run-time system. (Of course, this imposes new constraints on those techniques and opens up new opportunities for them to improve the code, such as propagating constants that cannot be known statically but are easily determined at run time.) This diaspora of compiler technology will break down our traditional notions of the various tools’ roles & responsibilities.

We agreed that finding the right structure for these new tool sets will require experimentation with different designs and approaches.

We agreed that we need new names for the tools in this world, to free all of us from our preconceived notions of their roles.



Why today?

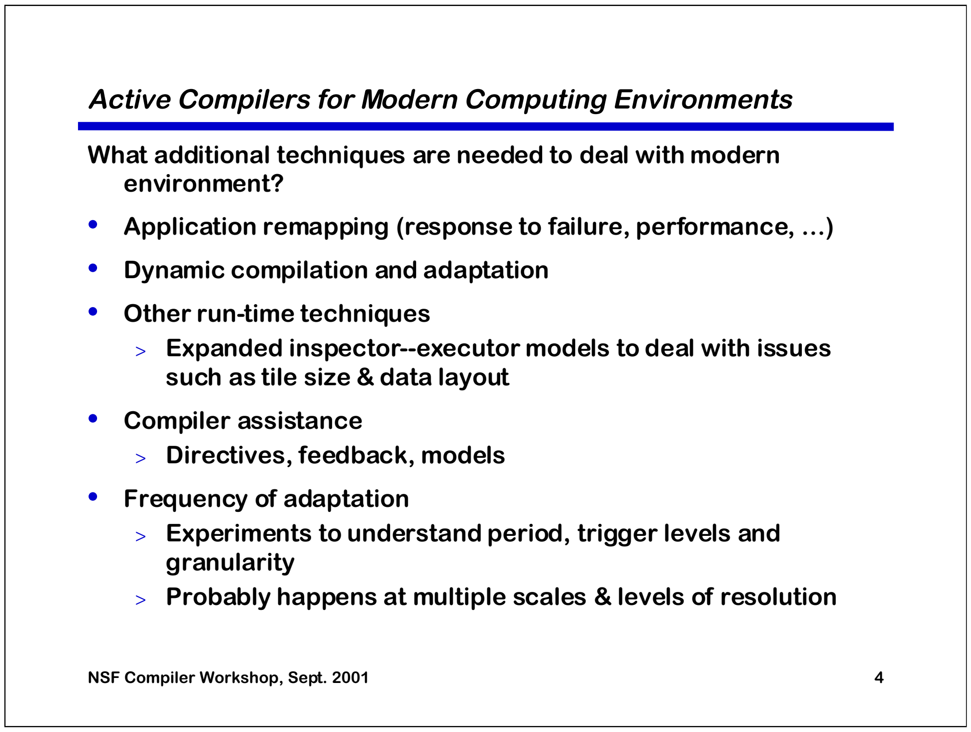
- Architectural change is accelerating
  - Transmeta, Itanium
- Execution environment is changing
  - Dynamic, distributed, heterogeneous world is here
  - It's getting even more so …
- Translation environments are emerging
  - Jalapeno, HotSpot, Dynamo, Microsoft's CLR

NSF Compiler Workshop, Sept. 2001 3

This is almost a throwaway slide. However, it captures part of the discussion that did not make it onto the flip chart.

Why is this work important today? Developments in the real world of computing (rather than the academic research lab) are creating a complex world where traditional program preparation and execution techniques make less sense. We see the emergence of new, more complex architectures, such as the Transmeta machine and the Itanium -- both with significant interest in the commodity microprocessor market. We see people trying to use the dynamic, heterogenous world as a computing engine---from large scale Grid applications through automobiles (which typically have 10 to 15 processors connected via a network) through experiments using IP checksums to "steal" computation. We need paradigms for building, using, and maintaining software in this world.

One bright spot is the emergence of new translation environments that capture some of this dynamism.



Active Compilers for Modern Computing Environments

What additional techniques are needed to deal with modern environment?

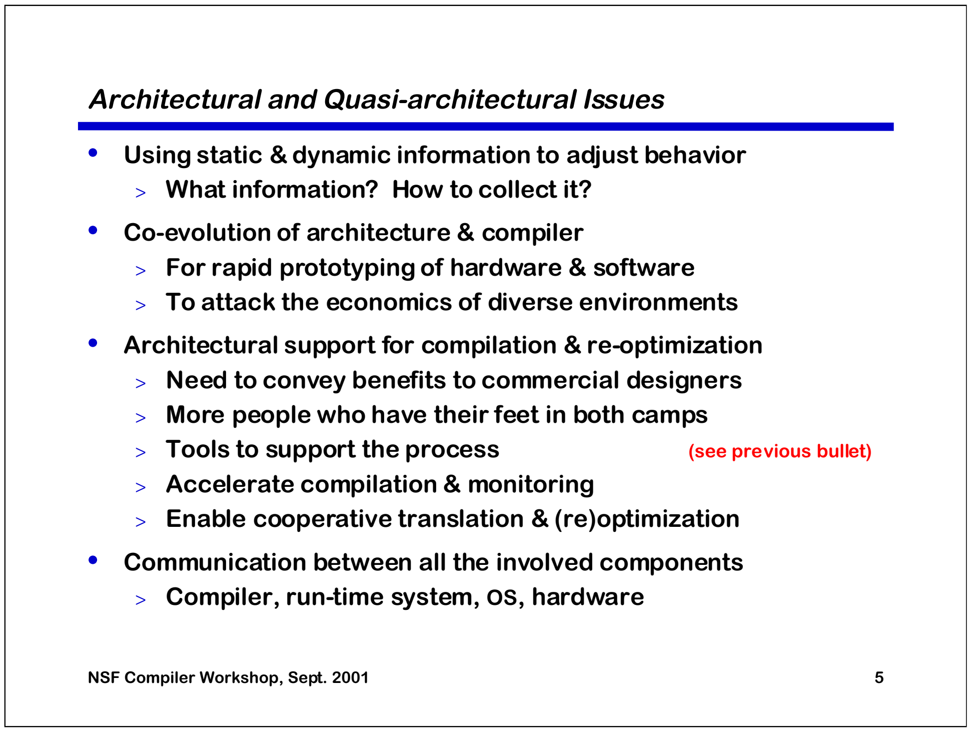
- **Application remapping (response to failure, performance, ...)**
- **Dynamic compilation and adaptation**
- **Other run-time techniques**
  - **Expanded inspector--executor models to deal with issues such as tile size & data layout**
- **Compiler assistance**
  - **Directives, feedback, models**
- **Frequency of adaptation**
  - **Experiments to understand period, trigger levels and granularity**
  - **Probably happens at multiple scales & levels of resolution**

NSF Compiler Workshop, Sept. 2001 4

The slide is fairly straight forward.

There are several cross-cutting issues that affect the entire discussion.

> Response to failure: In a dynamic environment, the application and its supporting run-time system must be prepared to deal with failure in the underlying infrastructure. This includes loss of communication as well as an actual hardware failure. Hard problems here include: detecting failure; checkpoint/restart capabilities (i.e.,where to write the checkpoints, how often, …); and migration.
>
> Dynamic (re-)compilation: If the tools are changing the code as it executes, it complicates any attempt to understand the active program state. Debugging, performance monitoring and modelling, and checkpointing all require, fundamentally, a map from source to state.

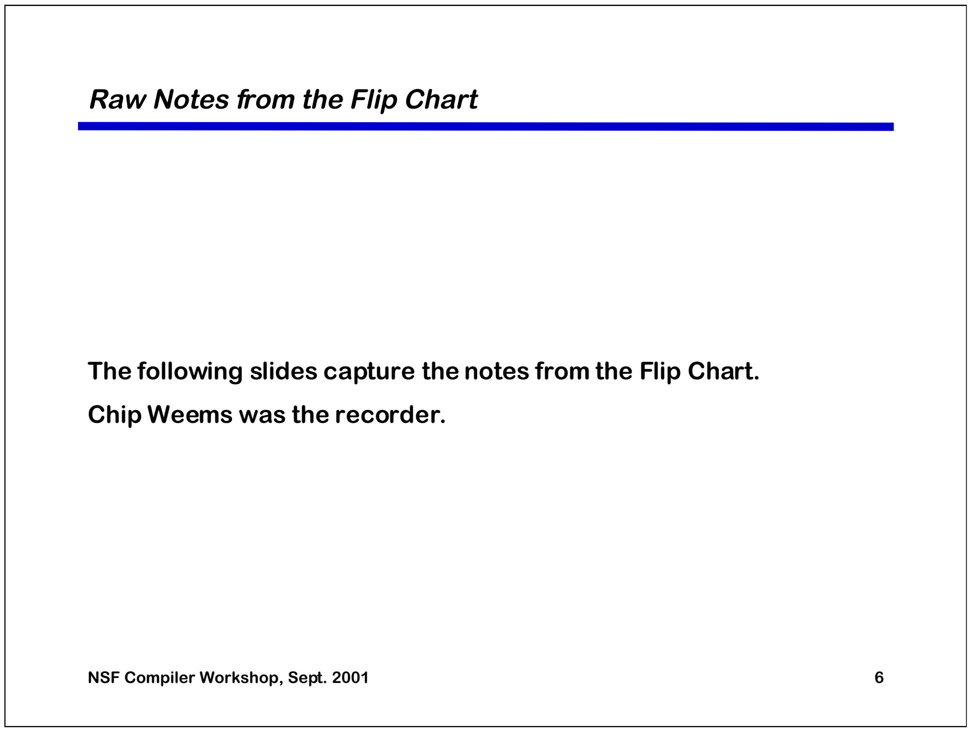


Architectural and Quasi-architectural Issues

- Using static & dynamic information to adjust behavior
  - What information? How to collect it?
- Co-evolution of architecture & compiler
  - For rapid prototyping of hardware & software
  - To attack the economics of diverse environments
- Architectural support for compilation & re-optimization
  - Need to convey benefits to commercial designers
  - More people who have their feet in both camps
  - Tools to support the process (see previous bullet)
  - Accelerate compilation & monitoring
  - Enable cooperative translation & (re)optimization
- Communication between all the involved components
  - Compiler, run-time system, OS, hardware

NSF Compiler Workshop, Sept. 2001 5

These were issues that provoked discussion, but were not directly related to the discussion of compilers for dynamic, heterogeneous environments.



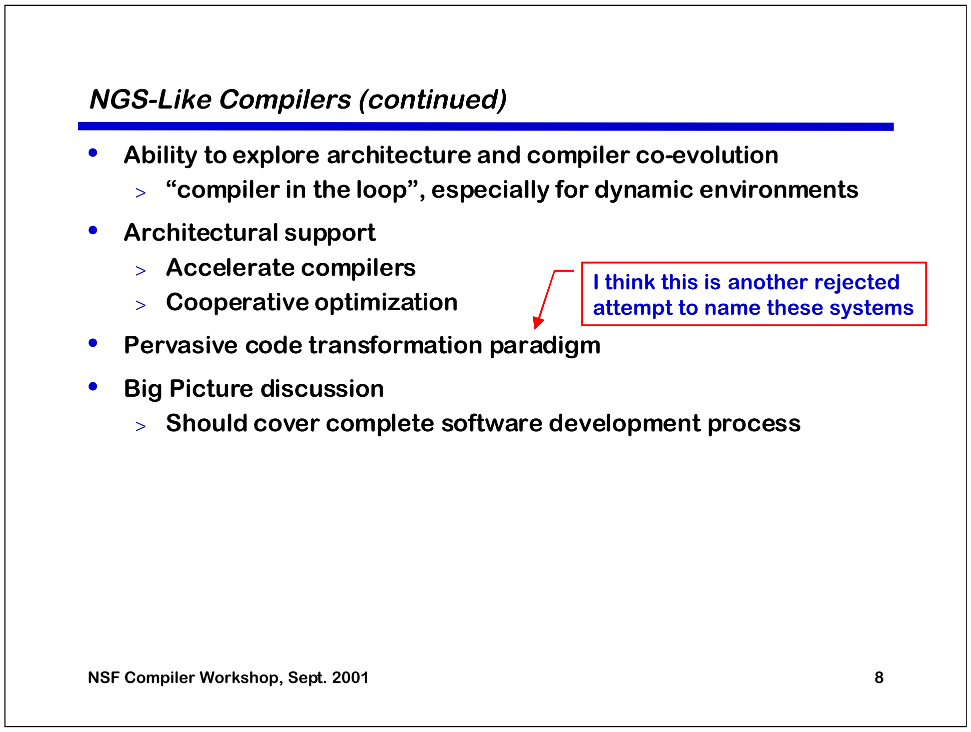
Raw Notes from the Flip Chart

The following slides capture the notes from the Flip Chart.

Chip Weems was the recorder.

NGS-Like Compilers

- What additional techniques are needed?
  - > Remapping
  - > Dynamic compiler adaptation
  - > Compiler assists (from the user)
    - → Directives, feedback, & models
  - > Frequency of adaptation
- Using static & dynamic informaiton to adjust program behavior at run-time
- Communication between compiler, run-time, OS, and architecture to achieve “cooperative performance architecture”
- Agile compiler versus agility of our compilers



NGS-Like Compilers (continued)

- Ability to explore architecture and compiler co-evolution
  - “compiler in the loop”, especially for dynamic environments
- Architectural support
  - Accelerate compilers
  - Cooperative optimization
- Pervasive code transformation paradigm
- Big Picture discussion
  - Should cover complete software development process

I think this is another rejected attempt to name these systems

NSF Compiler Workshop, Sept. 2001 8

Big Picture Discussion

- Is it reasonable to speak of compilers, run-time, linking, or do these terms constrain our thinking
  - Need a new word
  - Proposed terms include continuous, active, …
- Performance models and approximations
  - Discussion on distinction between approximator & estimator
- Integrate run-time
- Enhanced inspector-executor approach
  - Fill in parameters for tiling, as an example
- Information flow among all the components
- Feedback loops among the components
- Empirical optimization
- Explore design space of <new term> through experimental studies of prototypes

Compiler-Architecture Research

- Interaction
  - More people who do both
  - Small integrated teams doing both
  - Tools that support the process
- Architectural support
  - Architects don't want to suffer legacy issues
  - Architects need convincing evidence of benefit
    - → Feeds back into the tools discussion
- Frequency of adaptation
  - Experiment with period, trigger level, granularity
- Compiler cognizant of architecture
- Liquid architectures
  - Compiler control of hardware, such as cache ways
- How do compiler and architecture researchers work together?